

[PATCH] [17/34] x86: PARAVIRT: Add a sched_clock paravirt_op

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg12043.html>

- From: Andi Kleen <ak@xxxxxxx>
- Date: Mon, 30 Apr 2007 17:49:49 +0200 (CEST)

From: Jeremy Fitzhardinge <jeremy@xxxxxxxx>
 The tsc-based get_scheduled_cycles interface is not a good match for Xen's runstate accounting, which reports everything in nanoseconds.

This patch replaces this interface with a sched_clock interface, which matches both Xen and VMI's requirements.

In order to do this, we:

1. replace get_scheduled_cycles with sched_clock
2. hoist cycles_2_ns into a common header
3. update vmi accordingly

One thing to note: because sched_clock is implemented as a weak function in kernel/sched.c, we must define a real function in order to override this weak binding. This means the usual paravirt_ops technique of using an inline function won't work in this case.

[This is against Andi's patch queue. It fixes the x86-64 build problem.]

Signed-off-by: Jeremy Fitzhardinge <jeremy@xxxxxxxxxxxxxx>
 Signed-off-by: Andi Kleen <ak@xxxxxxx>
 Cc: Zachary Amsden <zach@xxxxxxxxxxxx>
 Cc: Dan Hecht <dhecht@xxxxxxxxxxxx>
 Cc: john stultz <johnstul@xxxxxxxxxxxx>

```

---
arch/i386/kernel/paravirt.c | 2 -
arch/i386/kernel/sched-clock.c | 43 ++++++-----
arch/i386/kernel/vmi.c | 2 -
arch/i386/kernel/vmiclock.c | 4 +--
include/asm-i386/paravirt.h | 7 +++++-
include/asm-i386/sched-clock.h | 49 ++++++-----
include/asm-i386/timer.h | 2 -
include/asm-i386/vmi_time.h | 2 -
include/asm-x86_64/timer.h | 2 -
9 files changed, 78 insertions(+), 35 deletions(-)

```

```
=====  
Index: linux/arch/i386/kernel/paravirt.c  
=====
```

```
--- linux.orig/arch/i386/kernel/paravirt.c  
+++ linux/arch/i386/kernel/paravirt.c  
@@ -268,7 +268,7 @@ struct paravirt_ops paravirt_ops = {  
.write_msr = native_write_msr_safe,  
.read_tsc = native_read_tsc,  
.read_pmc = native_read_pmc,  
- .get_scheduled_cycles = native_read_tsc,  
+ .sched_clock = native_sched_clock,  
.get_cpu_khz = native_calculate_cpu_khz,  
.load_tr_desc = native_load_tr_desc,  
.set_ldt = native_set_ldt,
```

```
Index: linux/arch/i386/kernel/sched-clock.c  
=====
```

```
--- linux.orig/arch/i386/kernel/sched-clock.c  
+++ linux/arch/i386/kernel/sched-clock.c  
@@ -35,30 +35,9 @@  
* -johnstul@xxxxxxxxxxx "math is hard, lets go shopping!"  
*/  
  
-#define CYC2NS_SCALE_FACTOR 10 /* 2^10, carefully chosen */  
-  
-struct sc_data {  
- unsigned cyc2ns_scale;  
- unsigned unstable;  
- unsigned long long sync_base; /* TSC or jiffies at syncpoint*/  
- unsigned long long ns_base; /* nanoseconds at sync point */  
- unsigned long long last_val; /* Last returned value */  
-};  
-  
-static DEFINE_PER_CPU(struct sc_data, sc_data) =  
+DEFINE_PER_CPU(struct sc_data, sc_data) =  
{ .unstable = 1, .sync_base = INITIAL_JIFFIES };  
  
-static inline u64 cycles_2_ns(struct sc_data *sc, u64 cyc)  
- {  
- u64 ns;  
-  
- cyc -= sc->sync_base;  
- ns = (cyc * sc->cyc2ns_scale) >> CYC2NS_SCALE_FACTOR;  
- ns += sc->ns_base;  
-  
- return ns;  
- }  
-  
/*  
* Scheduler clock - returns current time in nanosec units.  
* All data is local to the CPU.  
@@ -79,7 +58,7 @@ static inline u64 cycles_2_ns(struct sc_
```

[PATCH] [17/34] x86: PARAVIRT: Add a sched_clock paravirt_op

```
* per CPU. This state is protected against parallel state changes
* with interrupts off.
*/
-unsigned long long sched_clock(void)
+unsigned long long native_sched_clock(void)
{
unsigned long long r;
struct sc_data *sc = &get_cpu_var(sc_data);
@@ -98,8 +77,8 @@ unsigned long long sched_clock(void)
sc->last_val = r;
local_irq_restore(flags);
} else {
- get_scheduled_cycles(r);
- r = cycles_2_ns(sc, r);
+ rdtscll(r);
+ r = cycles_2_ns(r);
sc->last_val = r;
}

@@ -108,6 +87,18 @@ unsigned long long sched_clock(void)
return r;
}

+/* We need to define a real function for sched_clock, to override the
+ weak default version */
+#ifdef CONFIG_PARAVIRT
+unsigned long long sched_clock(void)
+{
+ return paravirt_sched_clock();
+}
+#else
+unsigned long long sched_clock(void)
+ __attribute__((alias("native_sched_clock")));
+#endif
+
+/* Resync with new CPU frequency */
static void resync_sc_freq(struct sc_data *sc, unsigned int newfreq)
{
@@ -124,7 +115,7 @@ static void resync_sc_freq(struct sc_dat
because sched_clock callers should be able to tolerate small
errors. */
sc->ns_base = ktime_to_ns(ktime_get());
- get_scheduled_cycles(sc->sync_base);
+ rdtscll(sc->sync_base);
sc->cyc2ns_scale = (1000000 << CYC2NS_SCALE_FACTOR) / newfreq;
}
```

Index: linux/arch/i386/kernel/vmi.c

```
=====
--- linux.orig/arch/i386/kernel/vmi.c
+++ linux/arch/i386/kernel/vmi.c
```

[PATCH] [17/34] x86: PARAVIRT: Add a sched_clock paravirt_op

```
@@ -890,7 +890,7 @@ static inline int __init activate_vmi(vo
paravirt_ops.setup_boot_clock = vmi_time_bsp_init;
paravirt_ops.setup_secondary_clock = vmi_time_ap_init;
#endif
- paravirt_ops.get_scheduled_cycles = vmi_get_sched_cycles;
+ paravirt_ops.sched_clock = vmi_sched_clock;
paravirt_ops.get_cpu_khz = vmi_cpu_khz;

/* We have true wallclock functions; disable CMOS clock sync */
Index: linux/arch/i386/kernel/vmiclock.c
=====
--- linux.orig/arch/i386/kernel/vmiclock.c
+++ linux/arch/i386/kernel/vmiclock.c
@@ -65,9 +65,9 @@ int vmi_set_wallclock(unsigned long now)
}

/* paravirt_ops.get_scheduled_cycles = vmi_get_sched_cycles */
-unsigned long long vmi_get_sched_cycles(void)
+unsigned long long vmi_sched_clock(void)
{
- return vmi_timer_ops.get_cycle_counter(VMI_CYCLES_AVAILABLE);
+ return cycles_2_ns(vmi_timer_ops.get_cycle_counter(VMI_CYCLES_AVAILABLE));
}

/* paravirt_ops.get_cpu_khz = vmi_cpu_khz */
Index: linux/include/asm-i386/paravirt.h
=====
--- linux.orig/include/asm-i386/paravirt.h
+++ linux/include/asm-i386/paravirt.h
@@ -116,7 +116,7 @@ struct paravirt_ops

u64 (*read_tsc)(void);
u64 (*read_pmc)(void);
- u64 (*get_scheduled_cycles)(void);
+ unsigned long long (*sched_clock)(void);
unsigned long (*get_cpu_khz)(void);

/* Segment descriptor handling */
@@ -573,7 +573,10 @@ static inline u64 paravirt_read_tsc(void

#define rdtscll(val) (val = paravirt_read_tsc())

-#define get_scheduled_cycles(val) (val = paravirt_ops.get_scheduled_cycles())
+static inline unsigned long long paravirt_sched_clock(void)
+{
+ return PVOP_CALL0(unsigned long long, sched_clock);
+}
#define calculate_cpu_khz() (paravirt_ops.get_cpu_khz())

#define write_tsc(val1, val2) wrmsr(0x10, val1, val2)
Index: linux/include/asm-i386/sched-clock.h
```

```

----- /dev/null
+++ linux/include/asm-i386/sched-clock.h
@@ -0,0 +1,49 @@
+#ifndef _ASM_SCHED_CLOCK_H
+#define _ASM_SCHED_CLOCK_H
+
+/* Accelerators for sched_clock()
+ * convert from cycles(64bits) => nanoseconds (64bits)
+ * basic equation:
+ * ns = cycles / (freq / ns_per_sec)
+ * ns = cycles * (ns_per_sec / freq)
+ * ns = cycles * (10^9 / (cpu_khz * 10^3))
+ * ns = cycles * (10^6 / cpu_khz)
+ *
+ * Then we use scaling math (suggested by george@xxxxxxxxxx) to get:
+ * ns = cycles * (10^6 * SC / cpu_khz) / SC
+ * ns = cycles * cyc2ns_scale / SC
+ *
+ * And since SC is a constant power of two, we can convert the div
+ * into a shift.
+ *
+ * We can use khz divisor instead of mhz to keep a better percision, since
+ * cyc2ns_scale is limited to 10^6 * 2^10, which fits in 32 bits.
+ * (mathieu.desnoyers@xxxxxxxxxx)
+ *
+ * -johnstul@xxxxxxxxxx "math is hard, lets go shopping!"
+ */
+struct sc_data {
+ unsigned cyc2ns_scale;
+ unsigned unstable;
+ unsigned long long sync_base; /* TSC or jiffies at syncpoint*/
+ unsigned long long ns_base; /* nanoseconds at sync point */
+ unsigned long long last_val; /* Last returned value */
+};
+
+DECLARE_PER_CPU(struct sc_data, sc_data);
+
+#define CYC2NS_SCALE_FACTOR 10 /* 2^10, carefully chosen */
+
+static inline unsigned long long cycles_2_ns(unsigned long long cyc)
+{
+ const struct sc_data *sc = &__get_cpu_var(sc_data);
+ unsigned long long ns;
+
+ cyc -= sc->sync_base;
+ ns = (cyc * sc->cyc2ns_scale) >> CYC2NS_SCALE_FACTOR;
+ ns += sc->ns_base;
+
+ return ns;
+}

```

[PATCH] [17/34] x86: PARAVIRT: Add a sched_clock paravirt_op

```
+
+#endif /* _ASM_SCHED_CLOCK_H */
Index: linux/include/asm-i386/timer.h
=====
--- linux.orig/include/asm-i386/timer.h
+++ linux/include/asm-i386/timer.h
@@ -2,6 +2,7 @@
#define _ASMi386_TIMER_H
#include <linux/init.h>
#include <linux/pm.h>
+#include <asm/sched-clock.h>

#define TICK_SIZE (tick_nsec / 1000)

@@ -15,7 +16,6 @@ extern int no_sync_cmos_clock;
extern int recalibrate_cpu_khz(void);

#ifdef CONFIG_PARAVIRT
-#define get_scheduled_cycles(val) rdtscll(val)
#define calculate_cpu_khz() native_calculate_cpu_khz()
#endif

Index: linux/include/asm-i386/vmi_time.h
=====
--- linux.orig/include/asm-i386/vmi_time.h
+++ linux/include/asm-i386/vmi_time.h
@@ -49,7 +49,7 @@ extern struct vmi_timer_ops {
extern void __init vmi_time_init(void);
extern unsigned long vmi_get_wallclock(void);
extern int vmi_set_wallclock(unsigned long now);
-extern unsigned long long vmi_get_sched_cycles(void);
+extern unsigned long long vmi_sched_clock(void);
extern unsigned long vmi_cpu_khz(void);

#ifdef CONFIG_X86_LOCAL_APIC
Index: linux/include/asm-x86_64/timer.h
=====
--- linux.orig/include/asm-x86_64/timer.h
+++ linux/include/asm-x86_64/timer.h
@@ -1 +1 @@
-#define get_scheduled_cycles(x) rdtscll(x)
+#include <asm-i386/sched-clock.h>
-
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>